Remarks

Claims 1-23 are pending in the application. Claims 1-23 are rejected. Claims 1 and 11 are amended herein. All rejections are respectfully traversed. No new matter is added.

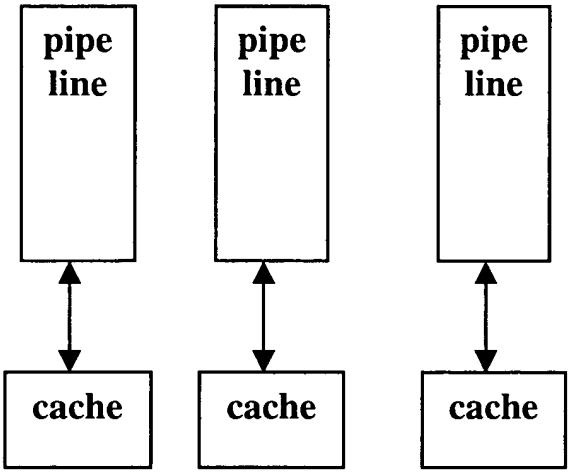
Claims 1 and 11 are amended herein. Claim 1 is amended to include limitations of claim 23 to clarify the structure of the serially connected stages of the pipeline, and the associated progressive cache. The amended claim does not change the scope of what is claimed, or present any limitations not previously presented. Claim 11 is amended to clarify data compression according to the invention. All amendments to the claims are supported in the specification. No new matter is added.

The invention renders objects. A rendering request describing an object to be rendered is defined. A progressive cache is queried to determine a cached element most representing a display image satisfying the rendering request. The cached element is sent to a starting stage of a rendering pipeline for the object, the starting stage associated with the cached element and an output of the starting stage is sent to an input of a next stage of the rendering pipeline. The pipeline includes serially connected stage, and a cache associated with each stage. A final stage of the rendering pipeline determines the display image satisfying the rendering request.

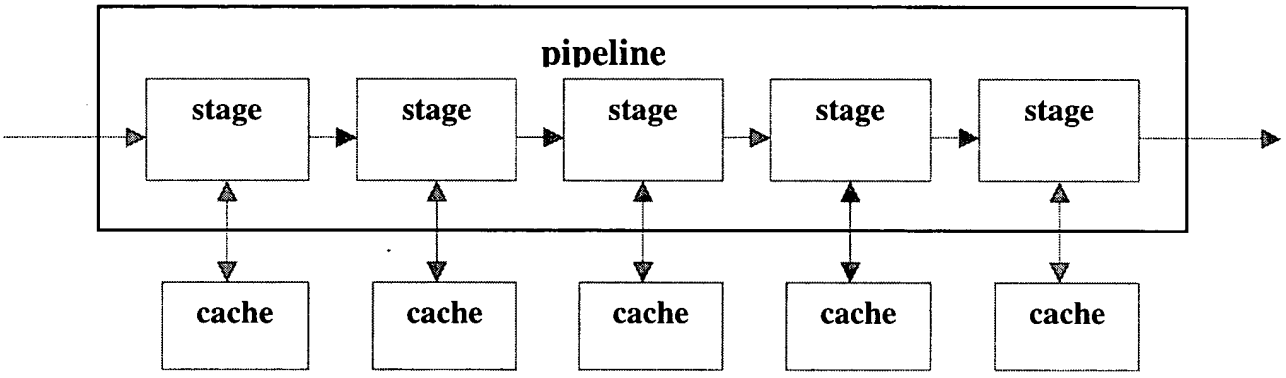
Claims 1-3 and 22 are rejected under 35 U.S.C. 102(b) as being anticipated by Hussain (U.S. Patent 6,801,203).

First, the structures of Hussain and what is claimed are entirely different.

Hussain has one cache for each one of multiple pipelines.



Claimed is one cache for each one of multiple, serially connected stage of a single pipeline.



Second, the processing in Hussain is entirely different than the progressive processing as claimed. From the structure above, it is clear that anything that is processed must be processed by the entire pipeline. In contrast, the progressive processing as claimed enables partial processing. The processing queries for the 'most finished' element in one of the caches arranged in a most finished to least finished order. Then, the most finished element is sent to a starting stage of the pipeline. Now, the staring stage does not need to be

the first stage. Rather, the starting stage is the next stage of the pipeline corresponding to the cached element, see claims. Therefore, with the progressive cache it is possible for an element to be processed only partially by the pipeline.

Hussain takes a series of queued rendering requests for a rendering pipeline and, according to the queued requests, pre-fetches unprocessed data to be rendered from a memory sub-system to a pixel queue for the entire pipeline. Hussain also searches the pixel queue to see if any of the requested unprocessed data is already there.

In contrast, the invention queries a progressive cache in response to a request to determine a most finished cached element representing a display image satisfying the rendering request. The progressive cache includes a plurality of caches arranged to store cached elements in a least finished to a most finished order, there being one cache associated with each stage. Therefore, instead of having to fetch unprocessed data from a memory subsystem or a pixel cache for processing by the entire pipeline as in Hussain, the invention can find cached elements that have already been processed by previous stages of the pipeline, and that can be inserted into a next stage of the pipeline to complete the rendering request. Hussain can never anticipate this.

Hussain does not describe the internal structure of the stages of his multiple pipelines. Certainly, Hussain does not describe one cache for each one of multiple serially connected stages of a single pipeline. Instead, Hussain, describes one cache for each one of multiple parallel connected pipelines.

Regarding claims 1, 22, and 23, the invention is directed to a method and apparatus for rendering using a progressive cache associated with stages of a rendering pipeline. Hussain never describes caches associated with stages of a rendering pipeline, as claimed. As stated above, Hussain's caches store unprocessed input for the entire pipeline, see col. 7, lines 13-25, above. In contrast, the invention queries a progressive cache to determine a cached element most representing a display image satisfying the rendering request. In other words, the invention looks in the progressive cache for an object that has been processed by at least one of the stages of the pipeline, which, if it were input back into the pipeline at the starting stage associated with the cached element, i.e., the next stage, it would match the rendering request at the output of the pipeline. Therefore, the output of the starting stage is sent to an input of a next stage of the rendering pipeline and so on, until a final stage of the rendering pipeline determines the display image satisfying the rendering request. There is no such thing described in Hussain. Hussain simply pre-fetches unprocessed data for the entire pipeline after looking at a request queue.

In claim 2, an output of a stage of the rendering pipeline is sent to the progressive cache. In contrast, the cache depicted in Figure 3 of Hussain is configured to store current and pre-fetched input data for the entire pipeline, see col. 6, lines 48-52, below:

> present invention is shown. Within the scope of the present invention, a graphics pipeline **301** not only utilizes a pixel cache **360** to cache incoming pixel data but also uses a pre-fetch FIFO **350** to enable pre-fetching of pixel data into pixel cache **360**. It is appreciated that graphics pipeline **301**

Nothing in Hussain even hints at sending output from a stage of a pipeline to a cache so that the cached data can be expediently processed by the next stage of the cache.

In claim 3, the progressive cache comprises a set of caches. As stated above, Hussain never describes a progressive cache storing cache elements for associated stages of a rendering pipeline. Further, the caches described by Hussain are not a progressive cache including a set of caches, as claimed. The pixel caches in Hussain are independent of each other. There is one pixel cache for each pipeline, see col 5, lines 49-61, below:

> Importantly, referring still to FIG. **2**, in one embodiment of the present invention, color pipeline **240** is coupled to a pixel cache **245**, and stencil z-buffer pipeline **250** is coupled to a pixel cache **255**. In this embodiment, pixel cache **245** is used for storing pixel-specific color data and is thus also referred to as color cache **245**. Likewise, pixel cache **255** is used for storing pixel-specific stencil z-buffer data and is thus also referred to as stencil z-buffer cache (s/z cache) **255**. Texture pipeline **260** is coupled to a texture cache **265** for storing texture data. Moreover, in an embodiment, each of color cache **245**, s/z cache **255** and texture cache **265** is independently coupled to a bus **280** for communicating information therewith.

The Examiner will note that Hussain asserts that it is important that the caches are independent of each other, and only associated with their respective pipelines.

Claims 4-10, 14-21 and 23 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hussain (U.S. Patent 6,801,203).

In claims 4-8, a particular cache in the set of caches is a preprocessed shape descriptor cache, a distance field cache, a distance values cache, an antialiased intensities cache, and a colorized image cache, respectively. As stated above with respect to claim 1, the progressive cache stores cache elements associated with stages of the pipeline. Hussain describes pixel caches, i.e., color, stencil z-buffer, and texture, that each store input data for

entire associated pipelines. The pixel caches of Hussain have nothing to do with pipeline stages.

The same is true for claim 9, where distance values for a component of a pixel of the display image are stored in the distance values cache. The distance values cache is associated with a stage of a pipeline, not an input for an entire pipeline as in Hussain.

In claim 10, the distance values for the component of the pixel of the display image are combined prior to determining an antialiased intensity for the component of the pixel. There is nothing in Hussain that could even hint at combining distance values prior to determining antialiased intensities for components of pixels. The Examiner's rejection is pure conjecture having absolutely no support whatsoever.

In claims 14-20, the rendering pipeline comprises a sequence of stages. Particular stage in the sequence of stages process the rendering request, determine a preprocessed shape descriptor, determine a distance field, distance values, antialiased intensities, and a colorized image, respectively. Further, in claim 21, the starting stage associated with the cached element is a next stage of a corresponding stage of a cache of the progressive cache containing the cached element. As stated above with respect to claim 1, Hussain never describes caches having cache elements associated with stages of a pipeline, as claimed. Hussain only caches unprocessed pipeline input data. Therefore, the Applicants respectfully request the rejection based on Hussain be reconsidered and withdrawn.

Claims 11-13 are rejected under 35 U.S.C. 103(a) as being unpatentable over Hussain (U.S. Patent 6,801,203) as applied to claim 1 and further in view of Cheng et al. (U.S. Patent 6,717,577).

Cheng describes a 3D graphics system where polygon vertex data is fed to a 3D graphics processor/display engine via a vertex cache used to cache and organize indexed primitive vertex data streams. The vertex cache fetches blocks of indexed vertex attribute data on an as-needed basis to make it available to the display processor.

The pipeline/cache structure of Cheng is like Hussain. There is only one cache for one entire pipeline, see below.

Cheng fails to cure the defects of Hussain. In particular, the vertex cache 212 cited by the Examiner is used only for input to the processing pipeline by the command processor 114, see Figure 1B, below:

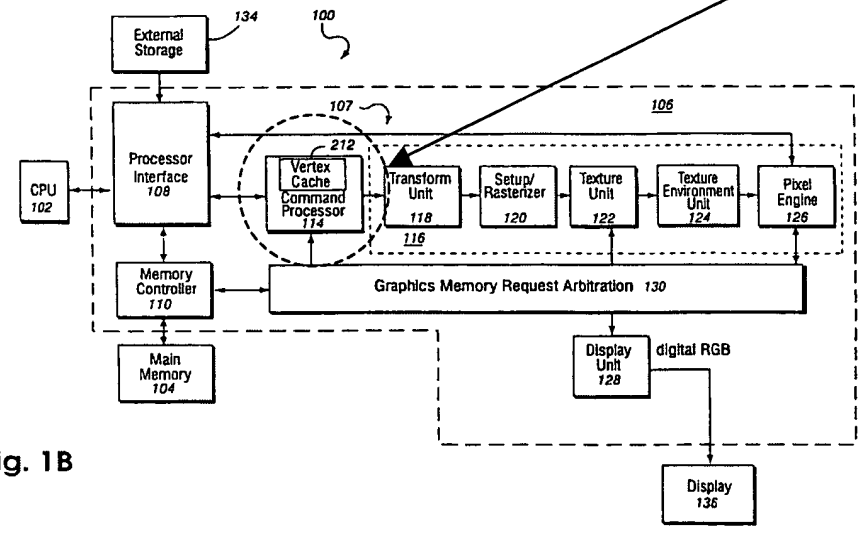


Fig. 1B

As stated above with respect to claim 1, the invention uses a progressive cache, which stores the output of pipeline stages. The progressive cache is queried to determine a cached element most representing a display image satisfying the rendering request. In other words, the invention looks in the progressive cache for an object that has been processed by at least one of the stages of the pipeline, which, if it were input back into the pipeline at the starting stage associated with the cached element, i.e., the next stage, it would match the rendering request at the output of the pipeline. Therefore, the output of the starting stage is sent to an input of a next stage of the rendering pipeline and so on, until a final stage of the rendering pipeline determines the display image satisfying the rendering request. A cache used only for input to a graphics pipeline as in Cheng can never make obvious what is claimed.

Claim 11 recites compressing data stored in a particular cache in the set of caches. Each cache according to the invention receives an output cache element of a corresponding stage of the processing pipeline and sends an input cache element to a next stage after the corresponding stage. Therefore, cache elements in the pipeline can be compressed according to the invention.

In contrast, Cheng's input-only vertex cache can receive compressed data, but the data must be decompressed before it is fed to the processing pipeline, see col. 3, lines 58-66, below:

> In accordance with a further aspect provided by this invention, the vertex data includes quantized, compressed data streams in any of several different formats (e.g., 8-bit fixed point, 16-bit fixed point, or floating point). This data can be indexed (i.e., referenced by the vertex data stream) or direct (i.e., contained within the stream itself). These various data formats can all be stored in the common vertex cache, and subsequently decompressed and converted into a common format for the graphics display pipeline. Such hardware support of flexible types, formats and numbers of attributes

Cheng must decompress a data stream before inputting it to the processing pipeline. The invention compresses data between stages of the processing pipeline. Cheng can never make this obvious.

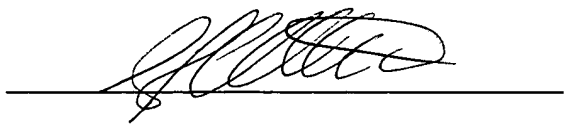
In claim 12, the progressive cache finds a cache element using hashing. As stated above, Cheng only describes an input cache for a processing pipeline. A person of ordinary skill in the art would never confuse Cheng's input cache with the claimed progressive cache having associated processing stages. Neither of the references describes or suggests, alone or in combination, a processing pipeline and progressive cache having cache elements associated with processing stages as claimed. Therefore, Cheng can never implicitly disclose finding a cache element using hashing.

The same is true for claim 13, where the progressive cache eliminates least recently used cached elements from a particular cache in the set of caches when the particular cache is full. Input caches as described in Hussain and Cheng can never make obvious the progressive cache, which stores cache elements associated with stages of the pipeline, as claimed.

It is believed that this application is now in condition for allowance. A notice to this effect is respectfully requested. Should further questions arise

concerning this application, the Examiner is invited to call Applicant's attorney at the number listed below. Please charge any shortage in fees due in connection with the filing of this paper to Deposit Account 50-0749.

Respectfully submitted,
Mitsubishi Electric Research Laboratories, Inc.

By 

Andrew J. Curtin
Attorney for the Assignee
Reg. No. 48,485

201 Broadway, 8th Floor
Cambridge, MA 02139
Telephone: (617) 621-7573
Customer No. 022199